COMPUTER AIDED SURFACE DISPLAYS

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
PAWAN KUMAR

to the
COMPUTER SCIENCE PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
JULY, 1980

CSP-1980-M-KUM-COM

CERTIFICATE

CERTIFIED that the thesis entitled 'Computer-Aided Surface Displays' has been carried out under my supervision and it has not been submitted elsewhere for a degree.

R. Sankar

Kanpur
July 1980

R. Sankar
Professor
Computer Science Programme
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

ACKNOWLEDGEMENT

I would like to thank Professor R. Sankar for his guidance and constant encouragement.

My thanks are due to Mr. H.K. Nathani for excellent typing and Mr. H.S. Tripathi for excellent cyclostyling.

Kanpur
July 1980

- Pawan Kumar

ABSTRACT

Algorithms for cross hatched and half toned representation of surfaces has been developed. Cross hatched representation algorithm is applicable to surfaces obtained by continuous linear transformation of, function of one variable. For this family of surfaces our algorithm is much faster than the already existing algorithms. For eliminating the hidden part of the surface we have modified the normally used Watkin's masking algorithm. Half-toned representation algorithm is applicable to surfaces of the form $Z = f(x,y)$. Here the rectangle in which the surface is defined is divided into subrectangles depending on the partial derivatives of z at left base corner of the rectangle then the points on the centre of the rectangles are projected on the view plane, to get the shadded picture of the surface.

CONTENTS

1. INTRODUCTION

This is a well known saying that "A picture is worth a thousand words", that this proverb holds today in the fields of physical sciences and engineering is demonstrated in the abundance of illustrations and figures contained in technical reports.

The two common methods of representing a surface is contour plots and planar projections. Contour plots depicts the qualitative properties of the surfaces and are useful, if numerical information is to be extracted from the figure. A planar projection depicts the surface in such a manner that its properties are most easily visualized. Further, planar projection has two common categories, cross hetched representation and continuous shadding or what is called a half tone represenation. In cross hatched representation mutually intersecting family of contours (normally one with fixed X-values, and second with fixed Y-values) are drawn to give the illusion of three dimensionality, whereas in half-tone representation points with varying density are plotted to give the 3-D view of the surface. Both the categories of projections have two kinds, orthographic projection where lines perpendicular to the view plane from the points on the surface are drawn which corresponds to the observer at infinite and perspective projection where line from the points on surface is joined to the point of observation, and in both the cases points or lines are drawn on the view plane where these set of lines intersects the plane. To give a proper 3-D view of the surface, those parts of it which are hidden from the observer are removed while drawing the surface.

We have developed algorithms for cross hetched and half tone representation surfaces with orthographic projection. As there are already many algorithms existing, there can not be any justification of giving a new algorithm without any significant improvement in the time or space compexity of them. The time complexity of the existing algorithms for cross hatched representation is dependent on both the complexity of the surfaces as well as the number of points considered on them, because if the surface is $z = f(x,y)$ the function f is evaluated for different values of x and y. Our algorithm is applicable to only a particular family of surfaces which are obtained using some continuous transformations on functions of one variable, for which we do away with the need of calculating (x,y,z) for different points chosen on the surface. We calculate (x,y,z) for points on only a particular curve lying on the surface and remaining points on the surface are obtained from these points using the transformation. This makes the time complexity of our algorithm independent of the complexity of the surface and depends only on the number of points taken on it. Moreover, if the equation of the surface is $f(x,y,z) = 0$, where z is not easily separable, then the existing algorithms require to solve the equation $f(x,y,z) = 0$ for fixed values of X and Y, making it even more complicated to determine (x,y,z), whereas our algorithm even though not applicable to any general $f(x,y,z) = 0$, generates many surfaces using transformations which can not be explicitly solved for z. (For example, see the surfaces in Figures 5 and 6.) The time complexity of our algorithm is dealt in Chapter 3. For the surfaces where our algorithm is applicable it is much faster than the existing algorithms.

For eliminating hidden lines we have modified the normally used Watkin's algorithm to make it faster and applicable to discontinuous curves.

-

2. MASKING ALGORITHMS (Hidden-line Elimination)

In general for a given surface and angle of view not all parts of the surface are visible, to get a proper view it is required to eliminate that part which is hidden from the observer. One of the commonly used algorithms for eliminating the hidden part is by Watkins, which takes a curve on the surface as input, eliminates the hidden parts of it and draws the rest. Our algorithm is a modification of it. Watkin's algorithm is briefly given below, for detail see Reference 5.

2-1. WATKIN'S ALGORITHM

The surface is scanned from front to rear. After proper rotations to get the required view it is projected on the view plane (place perpendicular to the line of view which in the case of orthographic view is an unique plane). A proper coordinate system is chosen on the view plane. Two arrays mask low and mask high are maintained, which respectively contains for each possible pen position across the page the minimum and maximum y values attained so far, for the points projected on the view plane. Mask low and mask high define two curves. That part of the segment which lies between these two curves would be hidden. This algorithm to see which part of the segment lies between the two masking curves, determines for each possible pen position across the page on the segment, the Y-values and is compared with that of corresponding mask high and mask low, if it lies between these values then this point would be hidden. And the pen is moved to this new point without drawing any line. Otherwise if it is not hidden then a straight line is drawn from the

previous pen position to this point and the masking arrays are updated. As the check is made at each and every pen position and accordingly line drawing routine is called to draw either a blank line or solid line, time required for masking as well as drawing the line is considerably large, consequently making the surface drawing rather slow.

2-2. MODIFICATION OF WATKIN'S ALGORITHM

We can do away with the need of checking for hidden point for each pen position with only a slight effect on the accuracy of hidden line elimination.

Let the line segment be given in an interval $[a,b]$, and the coordinates of the end points of the line segment be (a,Y_1) and (b,Y_2) respectively. The values of mask low and mask high at a and b are MLa, MLb, MHa and MHb respectively. Approximating that mask low and mask high vary linearly in the interval $[a,b]$, the intersection of line segment $[(a,y_1), (b,y_2)]$ with $[(a,MLa), (b,MLb)]$ and $[(a,MHa), (b,MHb)]$ is given below:

(1) Intersection with mask low:

$$I\,X1 = (MLa * Y_2 - MLb * Y_1) / (MLa - MLb + Y_2 - Y_1) \qquad (1)$$

$$I\,Y1 = a + (a-b) * I\,X1 / (Y_1 - Y_2) \qquad (2)$$

where (I X1, IY1) is the intersection point.

(2) Intersection with mask high -

$$I\,X2 = (MHa * Y_2 - MHb * Y_1) / (MHa - MHb + Y_2 - Y_1) \qquad (3)$$

$$I\,Y2 = a + (a-b) * I\,X_2 / (Y_1 - Y_2) \qquad (4)$$

where (IX2, IY2) is the intersection point.

The above equations holds when the following conditions are true -

(a) MLa - MLb + $Y_2 - Y_1 \neq 0$

(b) MHa - MHb + $Y_2 - Y_1 \neq 0$

(c) $Y_1 - Y_2 \neq 0$

When the above conditions hold then depending on the following cases part or whole of the line segment would be visible

(1) If a $\leq$ I X1 $\leq$ b and a $\leq$ I X2 $\leq$ b then the line segment intersects with both the masking arrays and its portion lying between I X1 to I X2 would be hidden.

(2.a) If a $\leq$ I X1 $\leq$ b and (I X2 $\leq$ a or I X2 $\geq$ b) then the line segment intersects with only mask low. In that case if MLa $\geq$ Y1 then the portion between I X1 to b would be hidden else the portion between a to I X1 would be hidden.

(2.b) If a $\leq$ I X2 $\leq$ b and (I X1 $\leq$ a or I X1 $\geq$ b) then the line segment intersects with only mask high and if MHa $\geq$ Y1 then the portion between IX2 to b would be hidden else the portion between a to I X2 would be hidden.

(3) If (MLa $\leq$ Y1 $\leq$ MHa and MLb $\leq$ Y2 $\leq$ MHb) or (Y1 $\geq$ MHa and Y2 $\geq$ MHb) or (Y1 $\leq$ MLa and Y2 $\leq$ MLb) then the whole line segment would be visible.

Cases when one or more of conditions (a), (b) and (c) are false:

(1) When only (a) is false -

The line segment $\overline{[(a,y_1), (b,y_2)]}$ would be parallel to the mask low segment. If $y_1 \leq$ MLa then the whole segment would be visible, else if a $\leq$ I X2 $\leq$ b then the analysis runs exactly as in the case 2.b above otherwise if Y1 $\leq$ MHa then the whole segment would be hidden.

(2) When only (b) is false, the analysis runs similar to the above case.

(3) When (c) is false -

line segment $[(a,y_1), (b,y_2)]$ would be parallel to the X-axis, I X1 and I X2 would be determined by the Equation (1) and (3) respectively. I Y1 and I Y2 would be equal to Y1. And the hidden line analysis would be same as the case when all the three conditions hold.

(4) When (a) and (b) are false -

If $MLa < Y_1 < MHa$ then the whole line segment would be hidden otherwise it would be visible.

These are the only four possible independent cases. Remaining combinations are covered in the cases discussed above. After drawing each segment of line the masking arrays are appropriately updated.

The above scheme for hidden line elimination works quite well when the masking curves are not very fast varying functions, otherwise also by choosing the interval $[a,b]$ smaller it can be made applicable.

The other problem with Watkin's algorithm is that it works only for continuous curves. Suppose that the partial function f is defined in the interval $[X1, X2]$, and is discontinuous or undefined in certain subintervals. If it is passed to Watkin's algorithm to draw, after eliminations the hidden part it would draw straight lines in the intervals where it is undefined. For example in drawing a hyperboloid it would connect its vertices by a straight line. The problem is solved very easily by first completing the partial function f, by assigning a large value η at all the points where itwas undefined. And the masking algorithm for each given x in the interval $[x_1, x_2]$ checks whether f(x) is equal to η, if it is equal to η then

no line is drawn from the previous point to this point. If f(x) is not equal to η but at the previous point the value of the function f was η then a blank line is drawn from the last point where f was not η to the present point. See Figure 4 (a hyperboloid intersecting with a cylinder).

2-3. RELATIVE TIME COMPLEXITY OF THE TWO MASKING ALGORITHMS

Suppose the function f is continuous on the closed interval $[x_1, x_2]$ and is approximated by M-1 line segments and the number of possible pen positions in the interval $[x_1, x_2]$ is N (N is normally much larger than M).

(1) Watkin's algorithm calculates the function f at all the possible pen positions N by linear interpolation and makes a check for hidden point. Let the time required to check whether a point is hidden or not be γ and to calculate the value of the function f at a point by linear interpolation by μ then the total time required for masking by Watkin's algorithm would be -

$$N * \gamma + (N - M) * \mu \qquad (5)$$

normally M is of the order of 100 and N about 1000. So the time complexity of Watkin's algorithm is O(N) which is independent of the number of line segment the curve is made of, which means that even a constant function (f(x) = c) will take as much time as any complicated function would take.

Moreover if making a call to draw a straight line takes π time (which for small line segment is independent of its length), then the the time required to draw the curve would be $\pi * N$ which is again $\Theta(N)$.

(2) In our algorithm (modification of Watkin's algorithm) intersection of each line segment is determined with both the masking arrays. If finding the intersection point and checking which part of the segment is above the masking function and which below, takes on average θ time then finding which part of the curve is hidden will require $2(M-1) * \theta$ time. (The factor 2 is because we find the intersection point with both the masking arrays). Moreover in updating the masking arrays we have to interpolate the function atmost at (N-M) pen positions. As each interpolation time is μ so the total time required by our algorithm is -

$$2(M-1) * \theta + (N-M) * \mu \qquad (6)$$

μ is usually much smaller than θ and γ.

Comparing Equations (5) and (6) we see that second term is common in both of them but the first term in Watkin's algorithm contains N where as our's M. As said before M is much less than N (N is atleast 10 times of M), also θ and γ are of the same order, so the second algorithm is several times faster than the first. Finally the time required to draw the curve in our algorithm is $\Pi * (M-1)$, which again is O(M) and much less than Watkin's algorithm.

3. SURFACE DISPLAY ALGORITHMS

In the following sections two different surface display algorithms are described. The first one draws cross hatched surfaces, which are obtained by continuous transformation of a function of single variable. The second one draws half tone picture of the surface of form $z = f(x,y)$.

3-1. DISPLAY OF A SURFACE OBTAINED BY GENERALIZED ROTATION OF A CURVE ABOUT A LINE

Generalised rotation as the name suggests is generalisation of the ordinary rotation. In rotation the distance of a point from the line of rotation remains constant, whereas in generalised rotation it varies with the angle of rotation. To be precise the generalised rotation about a line $\overline{r}$ and by an angle A is defined as follows:

Let IR be the set of real numbers, $ROT_{A,\overline{r}}$ the ordinary rotation transformations about $\overline{r}$ and by an angle A. Then the generalised rotation $G_{A,\overline{r}}$ is a linear transformation on IR^3 -

$$G_{A,\overline{r}} : IR^3 \longrightarrow I\,R^3$$

such that

$$G_{A,\overline{r}}(X) = f(A) * ROT_{A,\overline{r}}(X)$$

where $X \in IR^3$ and f is a continuous function from D_f a compact connected subset of R to IR such that $f(0) = 1$.

As can be easily seen ordinary rotation is a particular case of the generalised rotation when $f(A) = 1 \ \forall\ a \in IR$. If a function $y = H(x)$ is rotated about the X-axis using the generalised rotation function f then it can be easily verified that the equation of the surface generated would be -

$$Y^2 + Z^2 = (H(x)^2 * (f(\text{Arctan}(z/y)))^2 \qquad (1)$$

It is interesting to see that a plane surface can be obtained by the rotation of a line using the generalised rotation function –

$$f(A) = \text{Cosec}\,(A+P) * c$$

Assuming that the equation of the line is

$$Y = aX + b \qquad \text{or} \qquad H(x) = ax + b$$

and using Equation (1) to derive the equation of the surface generalised:

$$Y^2 + Z^2 = (a\,X + b)^2 * c^2\ \text{Cosec}^2(\text{Arctan}\ z/y + P)$$

or

$$(Y^2+Z^2) * \text{Sin}^2(\text{Arctan}(z/y)+P) = (a\,X+b)^2 * c^2$$

or

$$(Y^2+Z^2) * (\text{Sin Arctan}(z/y) * \text{Cos P} + \text{Cos Arctan}\ (z/y) * \text{Sin P})^2 = (a\,X + b)^2 * c^2$$

or

$$(Y^2 + Z^2) * (\text{Cos P} * Z / \sqrt{Y^2+Z^2} + \text{Sin P} * y / \sqrt{Y^2+Z^2}\ )^2 = (ax + b)^2 * c^2$$

or

$$z\ \text{Cos P} + Y\ \text{Sin P} = (ax + b) * c$$

which is equation of a plane !

The advantage of surfaces obtained by such generalised rotation is that there is an elegant way of cross hatched representation using two families of functions, where the calculation of (x,y,z) for the points on them is very easy and independent of the complexity of the surface. One of the family of functions is H(x) successively rotated (in the generalised sense) by an angle δA and projected on the X-Y plane

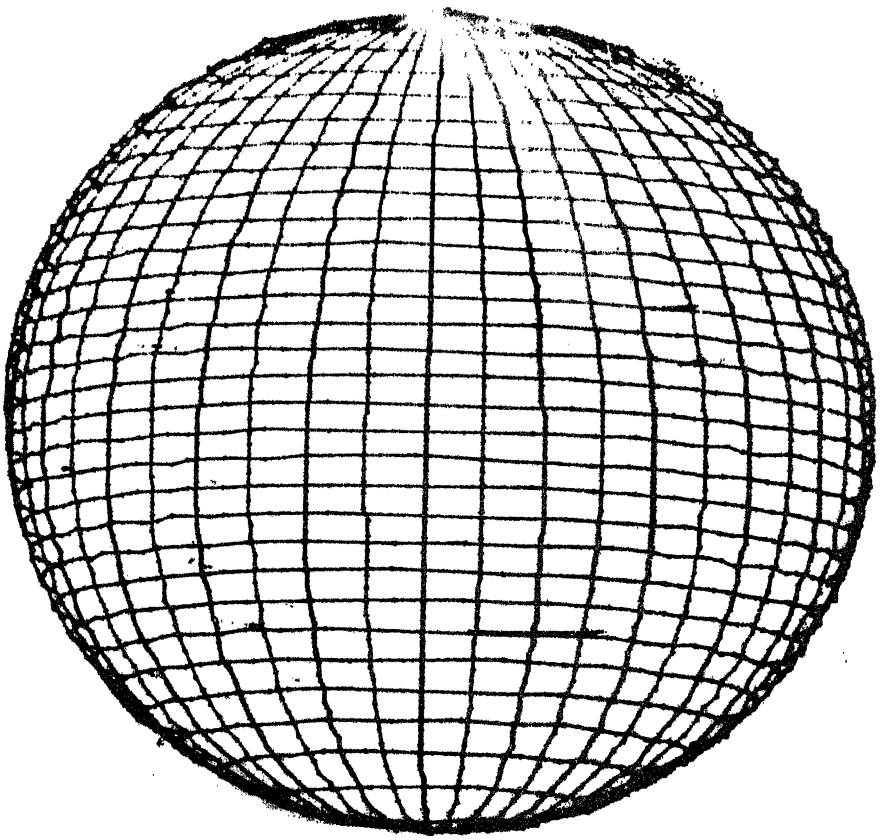
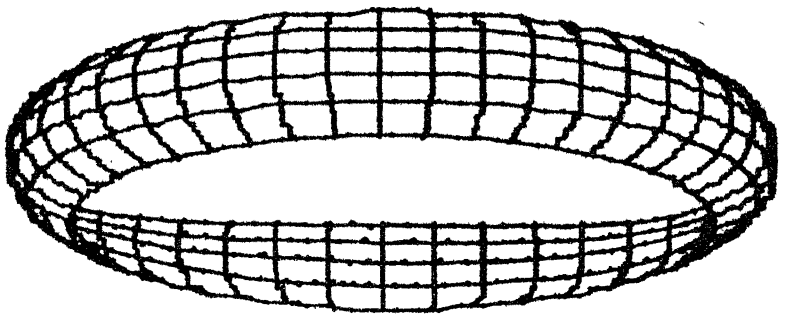


FIGURE 1: F(0)=1. NO. OF POINTS=4320
CPU TIME=12.69 SEC.

FIGURE 2: F(θ)=1.0
SURFACE OBTAINED BY THE ROTATION OF A SEMI CIRCLE. (SIMILER TO CYCLE TYRE)
NO. OF POINTS=1216, CPU TIME=3.91 SEC.



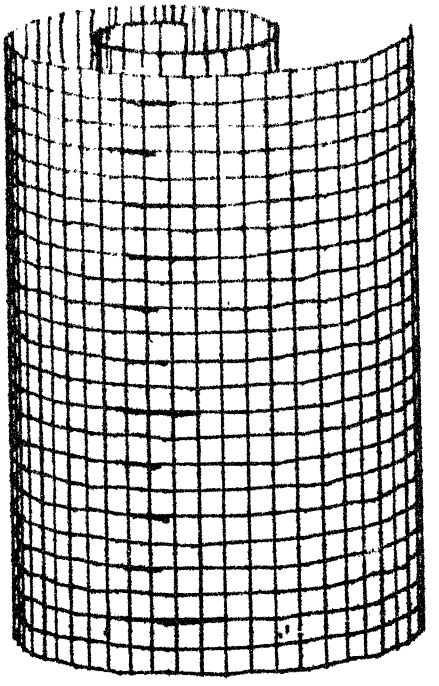


FIGURE 3: F(Ø)=EXP(-Ø/6ØØ)
NO. OF POINTS=2300. CPU TIME=6.57 SEC.

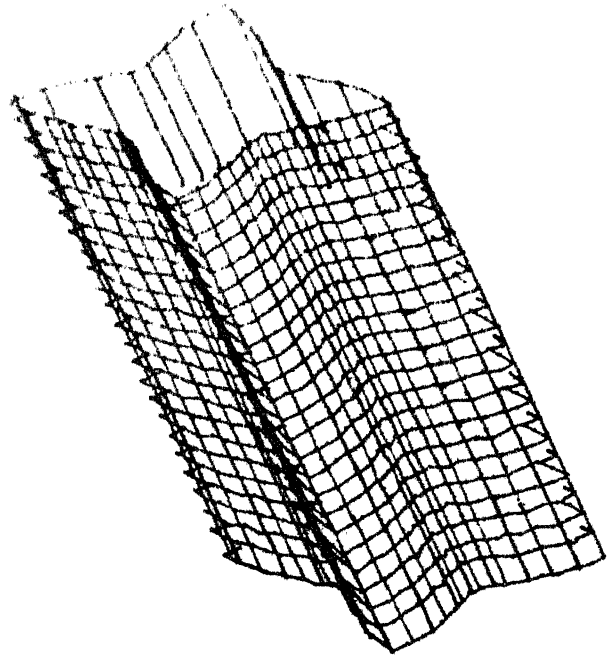


FIGURE 4: F(θ)=1-SIN(2θ)/2
NO. OF POINTS=2000. CPU TIME=6.5 SEC.

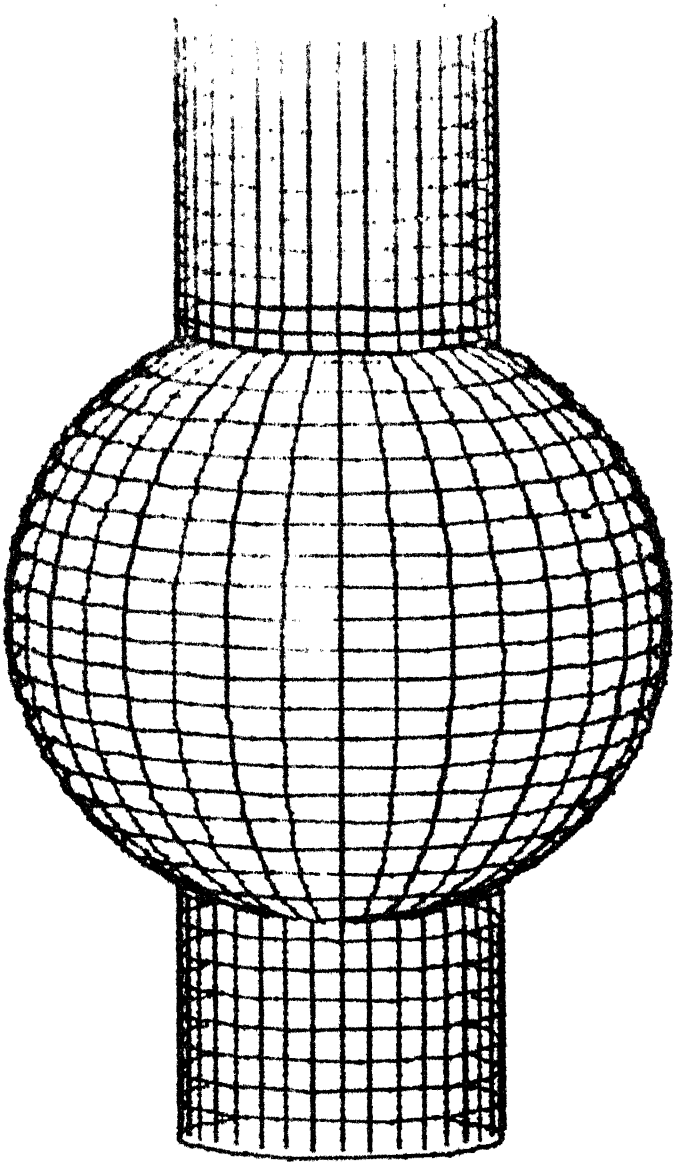


FIGURE 5: F(θ)=1.0. SPHERE AND CYLINDER
NO. OF POINTS=5200. CPU TIME=18.26 SEC.

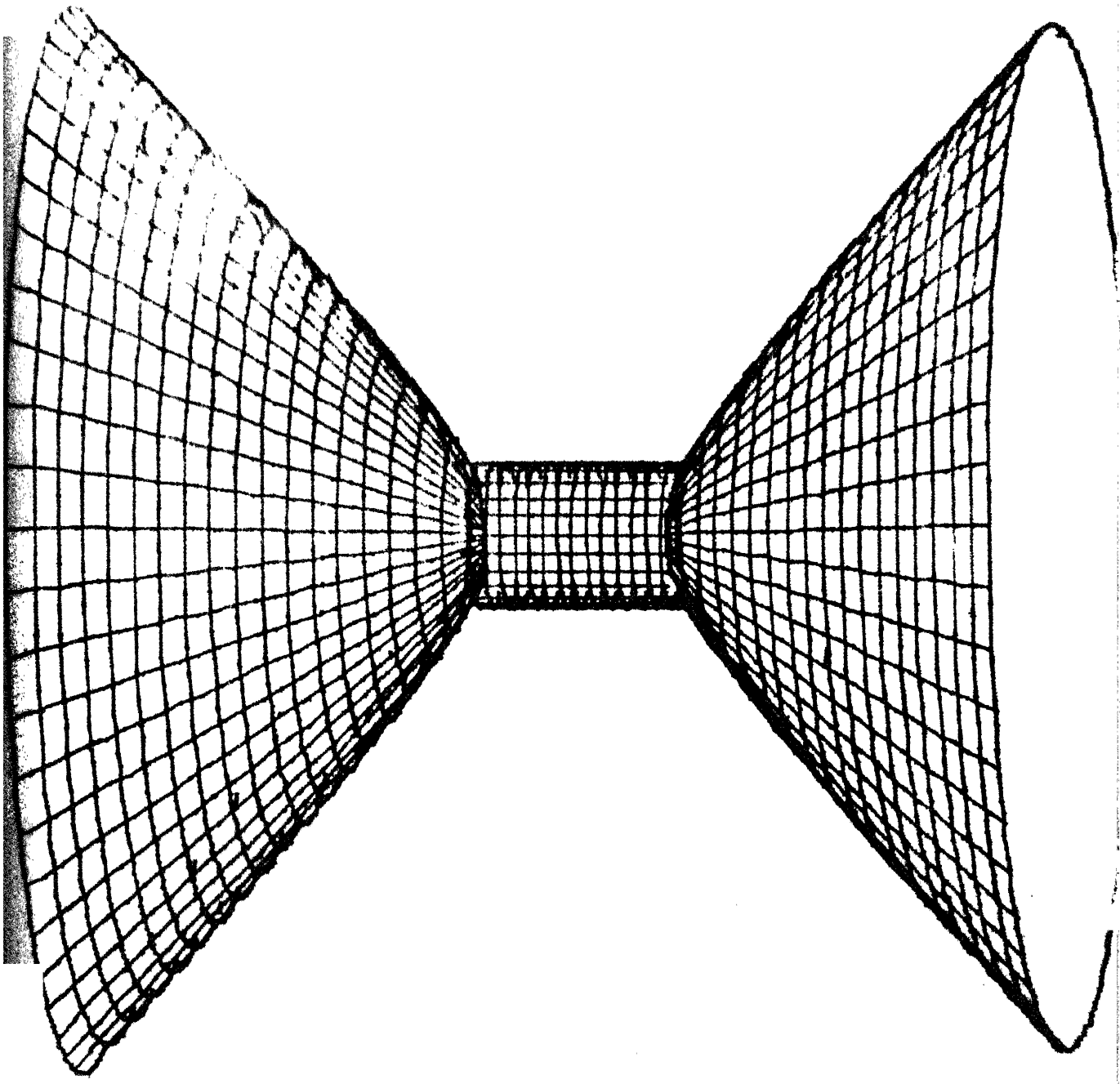


FIGURE 6: F(θ)=1.0. HYPERBOLOID AND CYLINDER
NO. OF POINTS=9427. CPU TIME=21.09 SEC.

FIGURE 7B

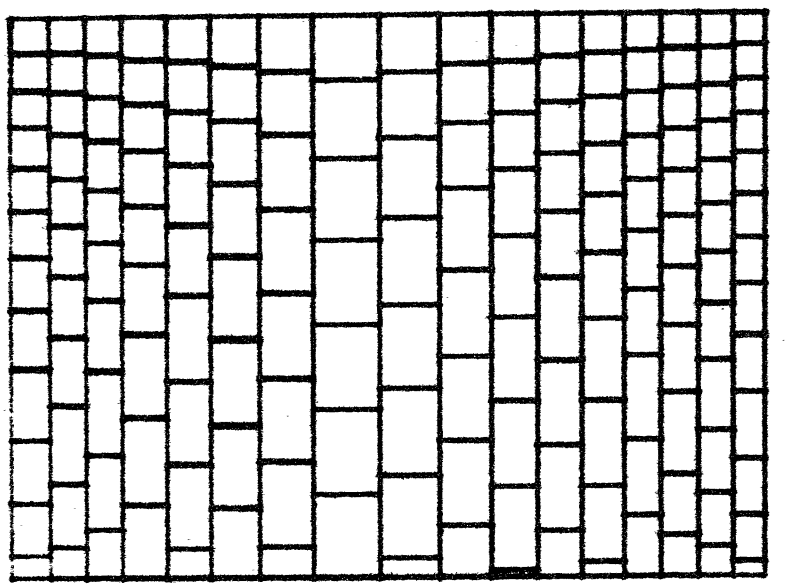


FIGURE 7A

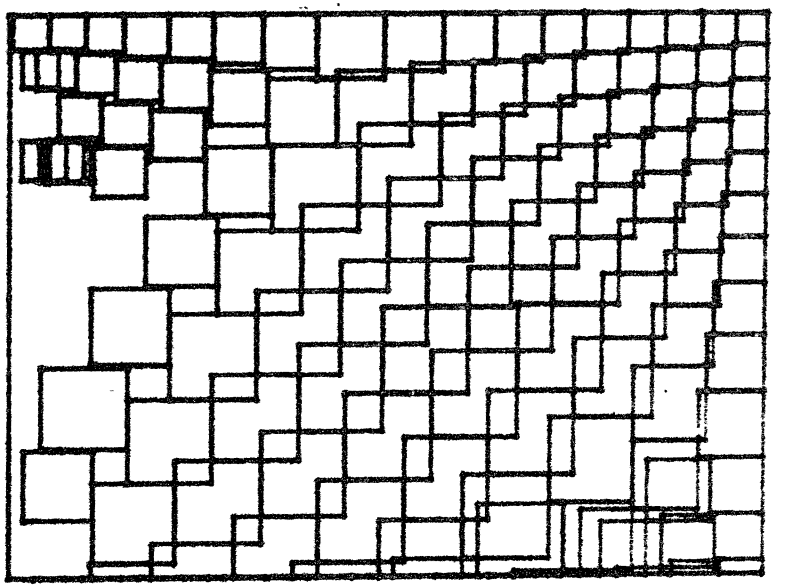


FIGURE 7C

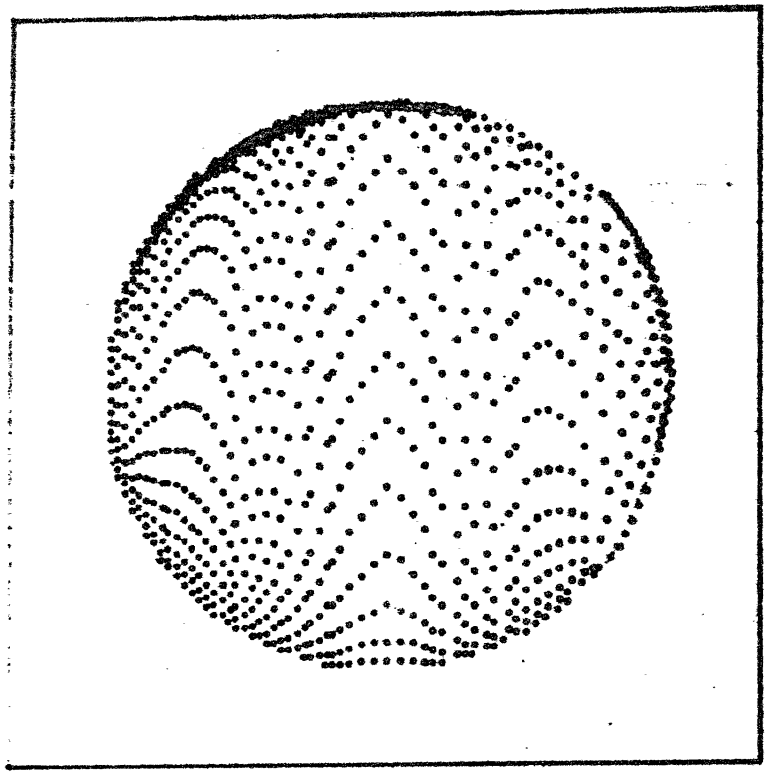


FIGURE 8: SPHERE

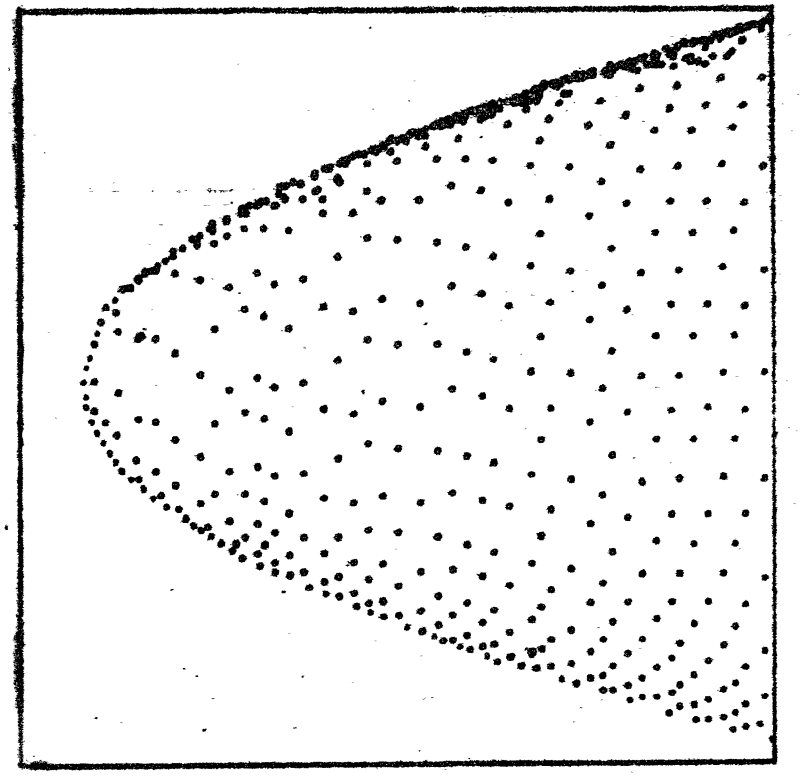


FIGURE 9: PARABOLOID

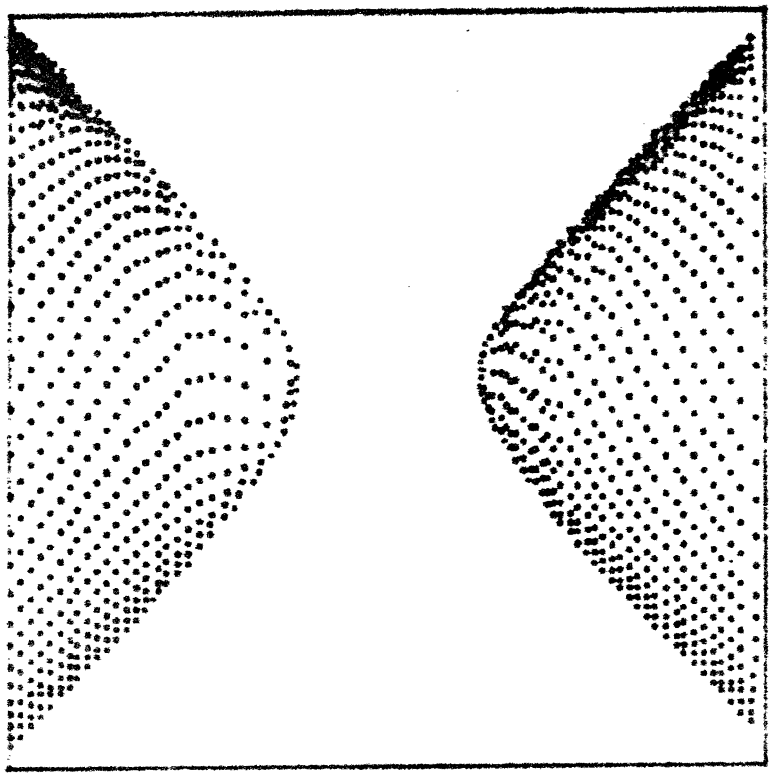


FIGURE 10: HYPERBOLOID

(assuming that the observer is in the direction of Z-axis at infinity in case the direction of observation is different from z then the surface is rotated to bring the Z-axis in the direction of obsertation), until it has been rotated by an angle of 360. The points on the unrotated function is calculated using Y = H(x) (z is zero for them), and the generalised rotation matrix $G_{\Delta,\gamma}$ is determined, which is successively applied to the function previously calculated, to give other members of the family. The second family consists of functions in planes parallel to the Y-Z plane lying at equal intervals in the domain of the function H(x), obtained by multiplying the generalised rotation function f(Δ) by H(x). Here again f(Δ) is determined, and all the functions of this family are obtained by multiplying H(x) (x is different for different members of the family) to the already calculated f(Δ). (In this case the transformation is one dimensional).

The general member of the first family is given by

$$F1_{\Delta}(i) = f(\Delta) * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \text{Cos}\Delta & \text{Sin}\Delta \\ 0 & -\text{Sin}\Delta & \text{Cos}\Delta \end{bmatrix} X \begin{bmatrix} x_i \\ H(x_i) \\ 0 \end{bmatrix} \qquad (2)$$

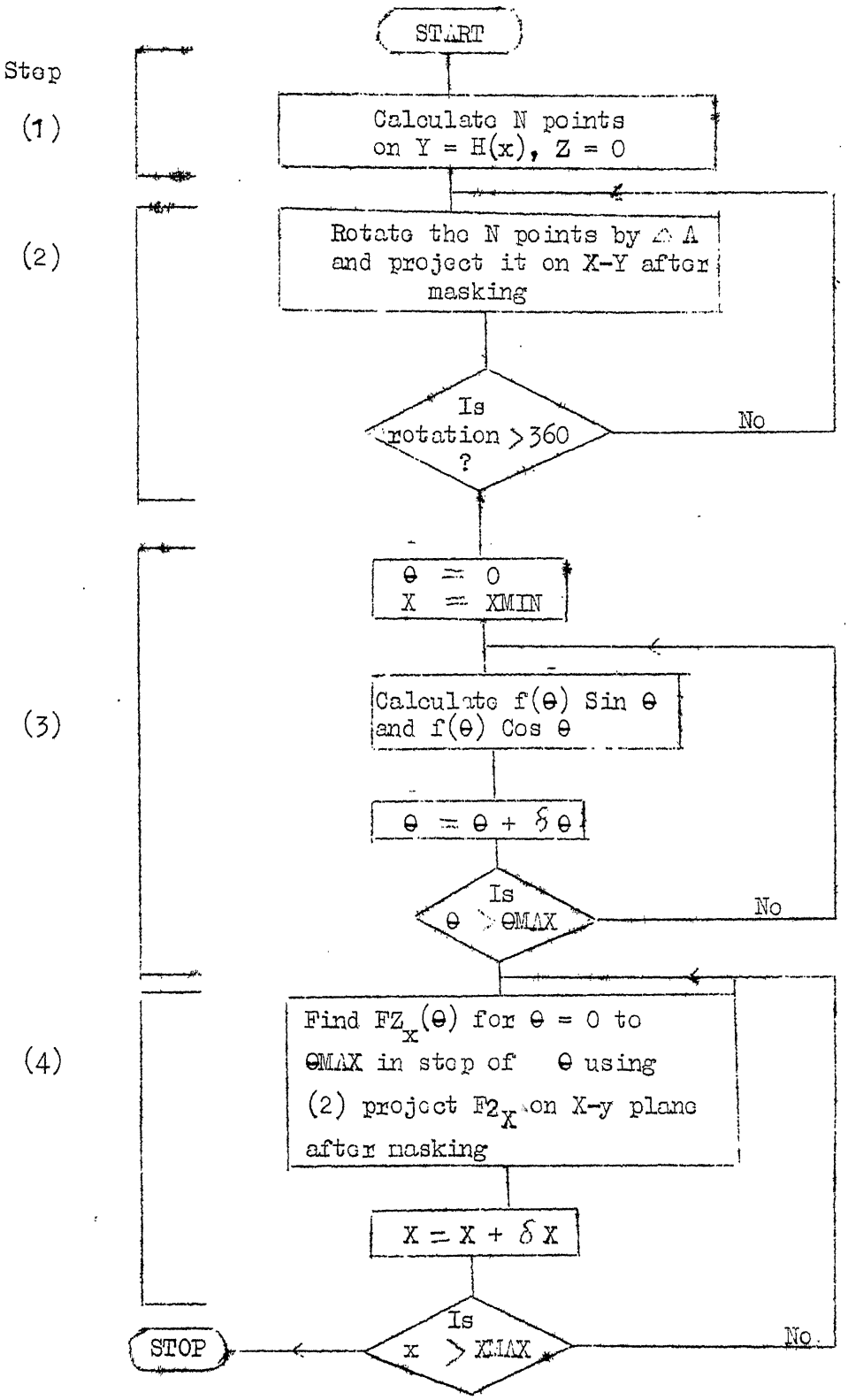
where $F1_{\Delta}(i)$ is the coordinate of the i^{th} point of that member of the family obtained by rotating H by angle Δ. x_i's and $H(x_i)$'s are same for all $F1_{\Delta}$.

The members of the second family are obtained by

$$F2_x(\theta) = \begin{bmatrix} x \\ H(x) * f(0) \text{ Cos } \theta \\ H(x) * f(\theta) \text{ Sin } \theta \end{bmatrix} \qquad (3)$$

These functions are passed to the masking routine where after proper rotation and elimination of the hidden part the functions are drawn.

The flowchart of the algorithm is given below:



The complete program is given in Appendix-1. We have plotted 6 surfaces using this algorithm. The total number of points taken on the surface and the CPU time is given with each figure. Figure 1 and Figure 2 are obtained by rotating a semi circle about X-axis with $f(A) = 1$. In the first case the centre of the circle lies on the X-axis whereas in the second it is at (0.1, 2.0), thus generating a sphere and a surface similar to that of a cycle tyre respectively. Surface 3 and 4 are obtained by rotating a straight line using the generalised rotation function $Exp(-A/600)$ and $(1+Sin\ 2A/2)$ respectively and the equation of surfaces 3 and 4 are given by:

$$Z = 600*Y*tan\ (-2\ Ln((Y^2 + Z^2)/2.3)) \text{ and}$$

$$(Y^2 + Z^2)^3 = 2.3(Y^2+Z^2 - ABS(YZ))^2 \text{ respectively.}$$

These are derived using Equation 1. As we see the above functions cannot be explicitely solved for Z, so if we try to plot them using conventional method the above two equations will have to be solved for each value of Y (fortunately they are independant of X). But our algorithm hardly requires any computation, and draws them in about 7 seconds.

Analysis of the time complexity of the algorithm

Consult the flowchart on Page 13 and the program in Appendix 1. Suppose we consider N points on the function H(x) and the time required to compute H(x) is t1. Then step one takes N*t1 time. In step two the N points calculated above are rotated successively by an angle of ΔA and projected on the X-Y plane. The rotated Y,Z coordinates are determined by multiplying the unrotated Y-coordinate of the N points by $f(A)\ Cos\ \Delta A$ and $f(A)\ Sin\ \Delta A$ respectively-so determining another

function of the family involves determing f(A), Sin ΔA, Cos Δ A and N+2 multiplications. If multiplications take time t2 then assuming that f(A), Sin Δ A and Cos Δ A all take time equal to M(x) i.e., t1, the determination of another function of the family requires t2(N+2)+3t1 time. The same time is required by all the successive functions. Hence the total time in Step 2 is:

$$(t2(N+2) + 3t1) * 360/\Delta A, \quad \text{let } M = 360/\Delta A$$

so the time in Step 2 is M(t2(N+2) + 3t1)

Step 3 requires (2t2 +3t1) * θMAX/$\delta\theta$, or , let MP = MAX/$\delta\theta$ then it is equal to MP(2t2 + 3t1).

The fourth step takes time = (t1+2t2*MP)*(XMAX-XMIN)/ΔX

let NP = (XMAX - XMIN)/ΔX, then the time equals to

NP(t1 + 2 t 2 * MP)

Step 1 and 2 determine the first family of functions and the total number of points on them is M*N, and the time required is Nt1 + M(t2(N+2)+3t1 Step 3 and 4 determine the 2nd family of functions with total number of points on them equal to MP*NP and the time required is MP(2t2 + 3t1) + NP(t1 + 2t2 * MP). Masking routine will take approximately 2θ(M*N+MP*NP) time (See Chapter 2) so the total time required is:

Nt1 + M(t2(N+2) + 3t1) + MP(2t2 + 3t1) + NP(t1+2t2 * MP) + 2θ(M*N+MP*NP)

or

t2 * M * N + 2t2 MP * NP + 2θ(M*N+MP*NP) + first order terms in M,N, MP and NP

as M,N, MP, NP are quite large so retaining only the second order terms in them, we see that it is proportional to M*N+MP * NP, i.e., the total number of points on the surface.

Interference of Rotation Surfaces

Figures 5 and 6 are interference of a cylinder with a sphere and hyperboloid of two sheet respectively. The program for single rotation surface is used with only a slight modification for two or more interfering rotation surfaces with a common line of rotation. In multiple rotation surfaces program, two masking arrays each are maintained. We take one surface at a time, both the families of this surface are drawn using different masking arrays. This is because no member of one family of functions masks any part of any member of the another family belonging to the same surface. And hence the same masking arrays cannot be used for both the families. After plotting one surface the corresponding two masking arrays (two mask low and two mask high) are combined to give the net masking effect of the surface plotted. Two copies of this resultant masking arrays are created. This would now be the base for plotting 2nd surface. By the very property of masking arrays that portion of the second surface would be eliminated which lies behind the first. So what we get essentially is projection of surfaces on view plane, with elimination of that portion of them which would not be visible to an observer at a particular angle of view.

3-2. HALF TONE REPRESENTATION OF SURFACE OF THE FORM $Z = f(x,y)$

Half tone pictures are shadded graphic displays. The general technique used in half tone display is to divide the rectangle in which the surface is defined, into nxn squares. Depending on the local property of the surface in each square a proper density of points are chosen and projected on the view plane. Size of the rectangle in these algorithms do not really depend whether the surface is constant

in certain region or fast varying, it is same in both the cases. The algorithm we have developed tries to see how the function f is varying, with X and Y and depending on that a proper rectangle size is chosen. The idea is to minimise the number of rectangles in the region where the surface is defined, which would enable to reduce the time required to draw the surface. Once we have divided the region in rectangles, we can have various ways of shadding it. We have used the easiest way, to put a point in the centre of the rectangle. In the region where the function $f(x,y)$ is slow varying we get a low density of points and where it is fast varying the points are dense, at the edges where the derivative, in the direction of observation is infinite we get a dense cluster of points marking the boundary of the surface.

Choosing the size of rectangles

Suppose that the surface $Z = f(x,y)$ is defined in the rectangle $x_0 <= X <= X1$, $Y_0 <= Y <= Y1$. Derivatives of Z with respect to x and y are calculated at (x_0,y_0), and the side of the square at this point is chosen to be $C/(ABS(\partial z/\partial x) + ABS(\partial z/\partial y))$ where c is a constant. We have taken it to be 0.015 in our program. Next the above procedure is repeated at $(x_0 + \delta x, y_0)$, where

$$\delta x = c/(ABS(\partial z/\partial x)_{x_0,y_0} + ABS(\partial z/\partial y)_{x_0,y_0})$$

and so on until we reach x1. This way we have got one row of non-interesecting squares, whose sizes are inversely proportional to the sum of absolute value of derivatives of Z with respect to x and y. The same procedure is repeated starting from x_0, and y equal to the height

already achieved at this X-value, until the whole region is covered by rectangles. This scheme as such leads to the problem of overlapping, one rectangle interferring with its neighbours (See Figure 7-A), and also some space being left uncovered between rectangles. This is avoided by reducing the length of the interfering rectangle to clear the overlapping (Figure 7-B). It solves the interference problem but at the cost of X-dependence of the length of the rectangles. Only the bottom row of rectangles have length depending on the derivations of the function f at that point, but the rectangles of the higher rows have length equal to that of the rectangle immediately below, unless the length of this rectangle is less than the rectangle below. So while we traverse from bottom to top the number of the rectangles in a row will either remain the same or increase but in no case will it decrease, even if the derivatives of the function decreases (see the Figure 7-B, the rectangles are put one above the other, their heights vary but length remains the same). To allow a certain degree of freedom in choosing the length of the rectangles we have to allow for certain amount of overlapping. The above scheme for interference has been modified to allow that the common area between overlapping rectangles divided by the area of the overlapped rectangle should not exceed 0.15. If it does exceed 0.15, the length of the rectangle is reduced to bring it down to 0.15. The effect of this row scheme is seen in Figure 7-C, length of the rectangles can be seen to vary in the X-direction also with the derivatives. For complete algorithm see the program in Appendix II.

Using this algorithm we have drawn sphere, paraboloid and hyperboloid of two seets, see Figures 8, 9 and 10.

The time complexity of our algorithm is dependent on the complexity of the surface.

4. CONCLUSION and SUGGESTION FOR FURTHER IMPROVEMENTS

The performance of cross hatched surface algorithm has been more or less what had been intended. The average CPU time for about 2500 points on the surface has been approximately 8 seconds, independent of the surface whereas Watkin's algorithm takes about 19 seconds with 2000 points for simple surface like Z = SinX * Sin Y / XY. For complicated surfaces like our Figures 3 and 4 it will take much more time, whereas using our algorithm they are drawn in only 6.57 and 6.5 seconds respectively.

The algorithm can be extended to accept multivalued H functions. We had restricted H to be single valued (see Section 3.1). This would require to modify the masking algorithm, because the present masking program keeps at the most two points for one pen position (they correspond to mask high and mask low), more than that for a particular pen position would be eliminated. So far multivalued functions only two of its single valued ranches would be kept rest of the portion would be eliminated even though it may be visible.

Secondly the family of surfaces which can be put in the form $z^2 + Y^2 = H^2(x)g^2(ARCTAN(z/y))$ may be extended to include those functions which do not belong to this family but their inverses belong to it. For example the surface Z = 1/X-1/Y cannot be put in the above form but 1/z = 1/X-1/Y belongs to the family of functions acceptable to our algorithm. To accept this extended family the algorithm requires only a minor modification. All the steps are carried out exactly as before, only at the time of actually drawing the surface Z-values are inverted.

The half toned surface display algorithm has not really been a success, as the shadded surface does not give a good 3-D view of the surface. This algorithm is not supported by masking and rotation. Consequently we can have only one view of the surface, that is observer on Z-axis at infinite (even the cross hatched surface display for this angle of view for paraboloid and hyperbolid does not very much reveal the properties of them). Secondly for shadding we did not have any hardware support, so the dimension and intensity of the points could not be controlled only the density could be varied, without which it is difficult to get a good 3-D view of the surface.

Possibly a better algorithm for getting shadded surfaces could be to choose points on the surface depending on the curvature (and not derivative) of the surface in its neighbourhood and then properly projecting it on the view plane.

REFERENCES

1. Levin, J.Z., "A parameteric algorithm for drawing pictures of solid objects composed of quadric surfaces" CACM, Vol. 19, No. 10 (1976).

2. Griffiths, J.G., "A surface display algorithm", CAD, Vol. 10, No. 1, (Jan. 1978).

3. Griffiths, J.G., "A data structures for the elimination of hidden surface by patch subdivision", CAD, Vol. 7, No. 3 (July 1975).

4. Butland, J., "Surface drawing made simple" CAD, Vol. 11, No. (Jan. 1979)

5. Watkins, S.L., "Masked three dimensional plot program with rotations", CACM, Vol. 17, No. 9 (Sept. 1974).

6. Robert, M., "Visible surface algorithm for quadric surfaces", IEEE Trans on Compt., 1972, pp. 1.

7. Ruth, A. Weiss, "Orthographic view of combination of plane and quadric surfaces", JACM, 1966, pp. 144.

8. Robert, B.R., "Perspective representation of functions of two variable", JACM, Vol. 15, No. 2 (April 1968).

9. Jack, W., "A procedure for generation of three dimensional half-toned computer graphics representation", CACM, Vol. 13, No. 9 (Sept. 1970)

10. Gourand, H., "Continous shadding of curved surfaces", IEEE Trans on Compt., June 1971, pp. 623.

```
                    ******************
                    *                *
                    *   APPENDIX 1   *
                    *                *
                    ******************

        PROGRAM FOR DRAWING GENERALISED ROTATION SURFACES


DIMENSION MASK(2000),Z(200),W(4),Y(200)
DIMENSION PY(100),PZ(100),PX(200)
INITIALIZATION OF DIFFERENT VARIABLES
-------------------------------------------------------------------
NUM: IS THE NUMBER OF POINTS ON THE CURVE F(X) , TAKEN TO PLOT
     CIRCLES OF ROTATIONS.
DELPHI: IS THE ANGLE IN DEGREE, BY WHICH THE CURVE F(X) IS
        ROTATED SUCCESSIVELY AND PROJECTED ON THE X-Y PLANE.
PHI,THETA: ARE THE ANGLES OF ROTATIONS BY WHICH THE WHOLE SURFA-
           ACE IS ROTATED, PHI BEING THE ROTATION ABOUT
        THE Y-AXIS; THETA ABOUT THE NEW X-AXIS.(ROTATED X-
          AXIS).
-------------------------------------------------------------------
DATA N/100/
DATA NUM/52/
DATA W/0.,1024.,0.,1024./
DATA DELPHI/8.0/
ACCEPT*,PHI,THETA
DATA SCALE/1.30/
DATA ANGLE/180.0/
IU=5
CALL INITDEV(IU)
CALL WINDW(W)
NPT=0
DRAWING THE FIRST FAMILY OF CURVES, OBTAINED BY SUCCESSIVE ROTATIO
-------------------------------------------------------------------
DO 10 NPOINT=1,61
X=(NPOINT-1)*0.015
Z(NPOINT)=0.0
TEM=F(X)
IF (TEM .EQ. 10000.0) GO TO 10
NPT=NPT+1
PX(NPT)=X
Y(NPT)=TEM
XMAX=X
CONTINUE
 CALL SCAL(SCALE,0)
CALL XLAT(-30.0,-10.0)
ROTANG=0.0
NN=0 ; NN1=0
DO 30 J=1,N
DELP=(1.0-ABS(SIN(0.01745*ROTANG)))*DELPHI
IF (DELP .LT. 4) DELP=4.0
GRT=GROT(ROTANG)
COSPHI=COS(ROTANG*0.01745329)*GRT
SINPHI=SIN(ROTANG*0.01745329)*GRT
DO 20 I=1,NPT
PY(I)=Y(I)*COSPHI
PZ(I)=Y(I)*SINPHI
CONTINUE
NN=NN+1
MP=MP+NPT
CALL PLOT3D(111,PX,PY,PZ,4.0,4.0,1.5,NN,NPT,PHI,THETA,
1    4.5,4.5,10.,MASK)
ROTANG=ROTANG+DELPHI
IF (ROTANG .GT. ANGLE) GO TO 40
CONTINUE
CONTINUE
DRAWING THE SECOND FAMILY OF CURVES
-------------------------------------------------------------------
XINC=0.9/(NUM-1)*2.
X=0.0-XINC
NN=0
DO 100 I=1,100
X=X+XINC
IF (X .GT. XMAX) GO TO 150
IF ( (X+XINC) .GT. XMAX) X=XMAX
```

```
IF(TEM .EQ. 10000.0) GO TO 100
YI=TEM
IF(YI .LT. 0.015) GO TO 100
M=(YI/0.015)+1
M=M+M
MM=2*M
DO 50 J=1,MM+1
SI=(J-1)*180.0/M
CALL CIRCLE(YI,SI,YJ,ZJ)
Y(J)=YJ
Z(J)=ZJ
CONTINUE
NN=NN+1
XX=X*4.0
MP=MP+MM+1
CALL PLOT3D(11,XX,Y,Z,0.,4.0,1.5,NN,MM+1,PHI,THETA,
1   4.5,4.5,10.0,MASK)
IF (ABS(X-XMAX) .LT. 0.005) GO TO 150
CONTINUE
CONTINUE
CALL IDEN
CALL LINI(1024,1024,0)
CALL LINI(1024,1024,1)
TYPE*,MP
STOP
END
SUBROUTINE  CIRCLE(RADIUS,THETA,Y1,Z1)
R=RADIUS
THE=THETA
Y1=R*GROT(THE)*COS(THE*0.01745329)
Z1=R*GROT(THE)*SIN(THE*0.01745329)
RETURN
END
SUBROUTINE ROTPT(X,Y,THETA)
ROTPT rotates the picture about the point (X,Y) by an angle THETA
-----------------------------------------------------------------
CALL XLAT(-X,-Y)
CALL ROTAD(THETA,3)
CALL XLAT(X,Y)
RETURN
END
FUNCTION GROT(A)
Function GROT is the generalised rotation function
--------------------------------------------------
GROT=1.0
RETURN
END
```

Program for hidden line elimination

```
C**********************************************************************
      SUBROUTINE PLOT3D(IVXYZ,XDATA,YDATA,ZDATA,XSCALE,YSCALE,
     1          ZSCALE,NLINE,NPNTS,PHI,THETA,XREF,YREF,XLENTH,MASK)
C This routine is a modification of Watkin's masking algorithm.
C It accepts 3-dimensional data in various forms as input, rotate
C it in 3-space to any angle and plot the projection of the
C resultant figure onto the X-Y plane.
C Each call to PLOT3D causes one line of a figure to be ploted.
C PIPI is the number of plotter increament per inch.
C NYPI is the number of increaments available across the width
C of the page (Y-direction).
C Input parameters-
C IVXYZ is a four digit integer which is used to select various
C input options. Thease digits in decreasing order of magnitude
C would be refered to as X,Y and Z.
C When NLINE is equal to 1 it indicates the beginning of a new
C figure.
C NPOINTS is the number of points on this line.
C PHI and THETA are the two angles in degrees used to specify
C the desired 3-D rotation.
C XREF and YREF are the coordinates in inches relative to the
C plotter origin, to be used as the origin of the figure.
C XLENTH is the length in inches to which the plot is restricted.
C MASK is an array of 2*XLENTH*PIPI entries which is used to
C store the maskhigh and masklow. Maskhigh being stored in the
C odd numbered positions and masklow in even numbered positions.
C**********************************************************************
      INTEGER HIGH, OLDHI,OLDLOW
      DIMENSION XDATA(1),YDATA(1),ZDATA(1),MASK(1)
      DATA INIT,JVXYZ,SPHI,STHETA/-1,-1,-1.0E30,-1.0E30/
      IF(NLINE.EQ.0) GO TO 550
      IF(NLINE.NE.1) GO TO 20
      PIPI=100.0
      NYPI=1024
      I=NYPI+100
      CALL IPLOT(0,-I,3)
      LIMITX=XLENTH*PIPI+0.5
      I=LIMITX+LIMITX
      DO 10 K=1,I
         MASK(K)=INIT
   10 CONTINUE
      INIT=-1
      INCI=-1
      I=0
      IF(JVXYZ.EQ.IVXYZ) GO TO 70
      JVXYZ=IVXYZ
      INDZ=1
      INDY=1
      INDX=1
      INDV=1
      IF(JVXYZ.LT.1000) GO TO 30
      INDV=2
      JVXYZ=JVXYZ-1000
      IF(JVXYZ.LT.100) GO TO 40
      INDX=2
      JVXYZ=JVXYZ-100
      IF(JVXYZ.LT.10) GO TO 50
      INDY=2
      JVXYZ=JVXYZ-10
      IF(JVXYZ.LT.1) GO TO 60
      INDZ=2
      JVXYZ=IVXYZ
      IF(PHI.EQ.SPHI .AND. THETA.EQ.STHETA) GO TO 80
      SPHI=SIN(0.0174532925*PHI)
      CPHI=COS(0.0174532925*PHI)
      STHETA=SIN(0.0174532925*THETA)
      CTHETA=COS(0.0174532925*THETA)
      A11=CPHI
      A13=-SPHI
      A21=STHETA*SPHI
      A22=CTHETA
      A23=STHETA*CPHI
      SPHI=PHI
      STHETA=THETA
```

```
DO 530 K=1,NPTS
I=I+INCI
GO TO (90,100),INDX
X=XDATA(1)+(I-1)*XSCALE
GO TO 110
X=XDATA(I)*XSCALE
GO TO (120,130),INDY
Y=YDATA(1)+(I-1)*YSCALE
GO TO 140
IF (YDATA(I) .EQ. 10000.0) GO TO 135
GO TO 138
JY=10000
GO TO 530
IF (JY .EQ. 10000 .AND. YDATA(I) .NE. 10000.0) IFLAG=1
Y=YDATA(I)*YSCALE
GO TO (150,160),INDZ
Z=ZDATA(1)+(NLINE-1)*ZSCALE
GO TO 170
Z=ZDATA(I)*ZSCALE
XXX=A11*X+A13*Z+XREF
XX=XXX
IX=(XX*PIPI)
YYY=A21*X+A23*Z+YREF
YY=YYY+A22*Y
IY=(YY*PIPI)
IF(IX.LE.0) IX=1
IF(IX.GT.LIMITX) IX=LIMITX
IF(IY.LT.10) IY=10
IF(IY.GT.NYPI) IY=NYPI
IF(K.NE.1) GO TO 250
LOW=IX+IX
HIGH=LOW-1
MLOW=MASK(LOW)
MHIGH=MASK(HIGH)
IF(MHIGH-IY) 200,240,180
IF(MLOW-IY) 240,240,220
MASK(HIGH)=IY
IF(MLOW.EQ.-1) MASK(LOW)=IY
GO TO 240
MASK(LOW)=IY
CALL IPLOT(IX,IY,3)
JX=IX
JY=IY
GO TO 530
IF (IFLAG .NE.1) GO TO 255
CALL IPLOT(IX,IY,3)
IFLAG=0
JX=IX
JY=IY
GO TO 530
IF (IX.NE.JX) GO TO 260
GO TO 530
YINC=FLOAT(IY-JY)/ABS(FLOAT(IX-JX))
INCX=(IX-JX)/IABS(IX-JX)
MXP=JX+JX-1
LXP=MXP+1
IPLT=0
IF (MASK(MXP) .LT. JY) IPLT=2
IF (MASK(LXP) .GT. JY) IPLT=1
IPLT1=0
MX=IX+IX-1
LX=MX+1
IF (MASK(MX) .LT. IY) IPLT1=2
IF (MASK(LX) .GT. IY) IPLT1=1
IF (IPLT1 .EQ. 0 .AND. IPLT .EQ. 0) GO TO 455
IF (IPLT1 .EQ. 0 .OR. IPLT .EQ. 0) GO TO 360
IF (IPLT1 .NE. IPLT) GO TO 340
CALL IPLOT(IX,IY,1)
JXX=JX
IXX=IX
JYY=JY
IYY=IY
GO TO 380
IF (MASK(LXP) .GT. JY .AND. MASK(MXP) .LT. JY) IPLT=IPLT1
IF(MASK(LX) .GT. IY .AND. MASK(MX) .LT. IY) IPLT1=IPLT
IF(IPLT .EQ. IPLT1) GO TO 330
IFL=1
```

```
      IFL=)
      IT=IPLT+IPLT1
      A1=JY-IY
      IF(IT .EQ. 1) GO TO 365
      JXP=JX+JX-1
      IXP=IX+IX-1
      GO TO 367
  365 JXP=JX+JX
      IXP=IX+IX
  367 A2=MASK(JXP)-MASK(IXP)
      B1=IX-JX
      C1=FLOAT(IX)*FLOAT(JY)-FLOAT(JX)*FLOAT(IY)
      C2=FLOAT(IX)*FLOAT(MASK(JXP))
      DDD=B1*(A1-A2)
      C2=C2-FLOAT(JX)*FLOAT(MASK(IXP))
      IF(DDD .EQ. 0) GO TO 330
      IXI=B1*(C1-C2)/DDD+0.5
      IYI=(A1*C2-A2*C1)/DDD+0.5
      IF (IPLT .NE. 0) GO TO 370
      CALL IPLOT(IYI,IYI,3)
      CALL IPLOT(IX,IY,1)
      JXX=IXI
      IXX=IX
      JYY=IYI
      IYY=IY
      GO TO 380
  370 CALL IPLOT(IXI,IYI,1)
      JXX=JX
      IXX=IXI
      JYY=JY
      IYY=IYI
  380 IPLT=IPLT1
      LJ=MIN0(JXX,IXX)
      LI=MAX0(JXX,IXX)
      IF (IYY .GE. MASK(IXX+IXX-1) .AND. JYY .GE. MASK(JXX+JXX-1))
     1    GO TO 410
      IP=JXX-INCX
      DO 405 II=LJ,LI
      IP=IP+INCX
      IPP=IP+IP
      MASK(IPP)=JYY+YINC*(II-LJ)
  405 CONTINUE
      GO TO 450
  410 IP=JXX-INCX
      DO 420 II=LJ,LI
      IP=IP+INCX
      IPP=IP+IP-1
      MASK(IPP)=JYY+YINC*(II-LJ)
      IF (MASK(IPP+1) .EQ. -1) MASK(IPP+1)=MASK(IPP)
  420 CONTINUE
  450 CONTINUE
      IF (IFL .EQ. 1) GO TO 350
      CALL IPLOT(IX,IY,3)
      JX=IX
      JY=IY
  350 CONTINUE
      CALL IPLOT(JX,JY,3)
      I=I-1
      RETURN
      INIT=0
      RETURN
      END
*************************************
      SUBROUTINE IPLOT(IIIX,IIIY,JJ)
*************************************
      IF (JJ.EQ.3) IVIS=0
      IF (JJ.NE.3) IVIS=1
      CALL LINI(IIIX,IIIY,IVIS)
      RETURN
      END
```

```
                    ******************
                    *                *
                    *   APPENDIX II  *
                    *                *
                    ******************

        PROGRAM FOR GENERATING HALFTONE PICTURES


type
    LIST=record
                X,DX:real;
                Y,Dx:real
         end;
    A=record
            X,Y:real
      end;
    B=array[1..4] of real;
var
    OLDLIST,NEWLIST:array[1..400] of LIST;
    POINT:array[1..400] of A;
    W:array[1..4] of real;
    C,X0,X1,XMAX,Y0,Y1,YMAX,DEL,DELX,DELY,D1,D2,DELLOW,DELHIGH:real;
    JJ,I1,J:integer; FLAG:boolean;

    (* (X0,Y0), (X1,Y1) gives the rectangle in which the function
       is defined.
       DELLOW is the minimum rectangle size.
       DELHIGH is the maximum rectangle size.
       NEWLIST contains information (length and breadth), about the
       rectangles of the current row.
       OLDLIST contains information about the rectangles of the
       previous row.
       POINT contains the (X,Y) coordinates of the diagonal points
       of the rectangles of the present row. *)
procedure FIND(Xi:real;var Yi:real; var I:integer);
      (* Procedure FIND, finds the height achieved so far,
            for the gigen X=Xi *)
      var
         TEM:real;
      begin I:=1; TEM:=OLDLIST[1].X;
         while Xi>=TEM do
            begin I:=I+1;
                  TEM:=OLDLIST[I].X;
                  if TEM=0 then TEM:=1000.0
            end;
         if I<>1 then I:=I-1;
         Yi:=OLDLIST[I].Y+OLDLIST[I].DY;
      end;
procedure PRCNTOVRLP(X1,Y1,DEL:real;I:integer;var PRCNT:real);
      (* This procedure finds the percentage overlape of the given
            rectangles *)
      var
         AREA,DELAREA:real;
      begin
         with OLDLIST[I+1] do
            begin   AREA:=DX*DY;
                  DELAREA:=(X1+DEL-X)*(Y+DY-Y1);
            end;
         PRCNT:=DELAREA/AREA*100.0
      end;
procedure INTERFERE(var  X1,DEL,Y1,DELX:real;  I:integer);
      (* Procedure INTERFERE changes the side of the given rectangle
            to avoid interference *)
      var
         TEM,PRCNT:real;
      begin          DELX:=DEL;
         TEM:=OLDLIST[I+1].X;
         if (TEM<>0) and ((X1+DEL)>TEM) then
            begin
                  PRCNTOVRLP(X1,Y1,DEL,I,PRCNT);
                  if PRCNT>18.0 then DELX:=(TEM-X1)*(1.0-18.0/PRCNT)
                                          +18.0*DEL/PRCNT;
                  if PRCNT<0 then DELX:=TEM-X1;
            end
      end;
```

```
procedure INITDEV(I:integer);
    EXTERN FORTRAN ;
procedure WINDW(var W:8);
    EXTERN FORTRAN ;
procedure LINE(X,Y:real; I:integer);
    EXTERN FORTRAN ;
procedure XLAT(X,Y:real);
    EXTERN FORTRAN ;
procedure PLOT(X,Y:real);
      begin
          LINE(X,Y,0);
          LINE(X,Y,1);
      end;
procedure FRAME;
      begin LINE(X0,Y0,0); LINE(X1,Y0,1);
          LINE(X1,Y1,1); LINE(X0,Y1,1);
          LINE(X0,Y0,1)
      end;
function DFX(X,Y:real):real;
      (* function DFX and DFY are the derivative of the functio
            F, w.r.t. X and Y respectively *)
      var
          Z:real;
      begin
  Z:=(X-6.00)*(X-6.0)-(Y-6.0)*(Y-6.0)-0.25;
  IF Z>0 THEN DFX:=(X-6.0)/SQRT(Z)
         ELSE DFX:=1.0E20
      end;
function DFY(X,Y:real):real;
      var
          Z:real;
      begin
  Z:=(X-6.00)*(X-6.0)-(Y-6.0)*(Y-6.0)-0.25;
  IF Z>0 THEN DFY:=(Y-6.0)/SQRT(Z)
         ELSE DFY:=1.0E20
      end;
    (* MAIN PROGRAM BEGINS *)
begin
    X0:=4.0;X1:=8.0;XMAX:=8.0;
    Y0:=4.0;Y1:=8.0;YMAX:=8.0;
    DELLOW:=0.05;DELHIGH:=0.450;  READ(TTY,C);
    INITDEV(5);
    W[1]:=0.0; W[2]:=10.0; W[3]:=0.0; W[4]:=10.0;
    WINDW(W);
    XLAT(3.0,0.0);
    with OLDLIST[1] do
      begin
          X:=X0; DX:=X1-X0;
          Y:=Y0; DY:=0.0
      end;
    FRAME;
    X1:=X0; JJ:=1; FLAG:=true;
    while FLAG do
      begin J:=1;
          while X1<XMAX do
            begin FIND(X1,Y1,II);
                D1:=DFX(X1,Y1); D2:=DFY(X1,Y1);
                if (D1#1.0E20) and (D2#1.0E20) then
                  begin
                    if (ABS(D1)+ABS(D2))=0 then
                          DEL:=DELHIGH
                    else
                          DEL:=C/(ABS(D1)+ABS(D2));
                    if DEL<DELLOW then DEL:=DELLOW;
                    if DEL>DELHIGH then DEL:=DELHIGH
                    INTERFERE(X1,DEL,Y1,DELX,II);
                    if DELX>=0.1 then
                      begin
                          if (X1+DELX)>XMAX then DELX:=XMAX-X1
                          if (Y1+DEL)>YMAX then  DEL:=YMAX-Y1;
                          with NEWLIST[J] do
                            begin
                                X:=X1; DX:=DELX;
                                Y:=Y1; DY:=DEL;
                            end;
```

```
                                X:=NEWLIST[J].X+NEWLIST[J].DX/2.0;
                                Y:=NEWLIST[J].Y+NEWLIST[J].DY/2.0;
                                PLOT(X,Y)
                            end;
                        J:=J+1
                    end
                else
                    if J>1 then NEWLIST[J-1].DX:=NEWLIST[J-1].DX+DEL
                    X1:=X1+DELX
            end
        else X1:=X1+DELLOW;
    end;
(* END OF THE INNER WHILE LOOP *)
if J#1 then
  begin
      if OLDLIST=NEWLIST then FLAG:=false;
      for II:=1 to J do
        begin
            OLDLIST[II].X:=NEWLIST[II].X;
            OLDLIST[II].DX:=NEWLIST[II].DX;
            OLDLIST[II].Y:=NEWLIST[II].Y;
            OLDLIST[II].DY:=NEWLIST[II].DY;
        end;
      if JJ>J then for II:=J+1 to JJ do
                    with OLDLIST[II] do
                      begin    X:=0.0; DX:=0.0;
                               Y:=0.0; DY:=0.0
                      end;
      JJ:=J
  end
else for II:=1 to JJ do
          OLDLIST[II].DY:=OLDLIST[II].DY+DELLOW;
X1:=OLDLIST[1].X;
if (X1#X0) then
  begin
      X1:=X1-10*DELLOW;
      if X1<X0 then X1:=X0;
  end;
```